Research Update: April 5

Nirupama Tamvada

1. Figuring out simulation settings for Ambrogi et. al 2016

The cause-specific hazards of the outcome of interest and the competing risk follow proportional hazards models, specifically:

$$\alpha_{01} = 0.5t \exp(\beta_{01} Z)$$

$$\alpha_{02} = t \exp(\beta_{02} Z)$$

where both cause-specific hazards have the form of a Weibull distribution and a common set of covariates. Cause 1 is the one of interest with an incidence rate of 54 % while cause 2 has an incidence rate of 28 % with a common uniform censoring rate of $\sim 15\%$.

```
knitr::opts_chunk$set(cache = T)

# independent normal variables
p <- 20
n <- 400
rho <- 0.5

# Very sparse case
# Common betas for both competing risks
beta <- c(0.5, rep(0, 18), 0.5)
zero_ind1 <- which(beta == 0)
nonzero_ind1 <- which(beta != 0)

# Generate iid X's
X <- matrix(rnorm(p*n), nrow = n, ncol = p)
# XB matrix
suma <- X %*% beta
```

```r
# Function to generate survival times
create.times <- function(n, ch, sup.int = 100) {
  times <- numeric(n)
  i <- 1
  while (i <= n)
  { u <- runif(1)
  if (ch(0, -log(u)) * ch(sup.int, -log(u)) < 0)
  { times[i] <- uniroot(ch, c(0, sup.int), tol = 0.0001, y= -log(u))$root
  i <- i + 1
  }
  else {
    cat("pos")
  }}
  times
}

# binomial probability of cause 1
binom.status <- function(ftime, n, a01, a02, size = 1)
{ prob <- a01(ftime) / (a01(ftime) + a02(ftime))
out <- rbinom(n, size, prob)
out }


# Cause-specific proportional hazards
times <- vector()
f.status <- vector()
for (i in seq_len(n)) {
alpha.1 <- function(t) { ((0.5*t)*exp(suma[i])) }
alpha.2 <- function(t) { t*exp(suma[i]) }

cum.haz <- function(t, y) { stats::integrate(alpha.1, lower=0.001, upper=t,
                                             subdivisions=1000)$value +
 stats::integrate(alpha.2, lower=0.001, upper=t,
                 subdivisions=1000)$value - y }
times[i] <- create.times(1, cum.haz)
f.status[i]<- binom.status(times, 1, alpha.1, alpha.2) + 1
}

# Censoring
cens.times <- runif(n, 0, 6)
```

```r
# Censoring in status variable
f.status <- as.numeric(times <= cens.times) * f.status

prop.table(table(f.status))
```

```
f.status
     0      1      2 
0.1700 0.5325 0.2975 
```

```r
# times with censoring
times <- pmin(times, cens.times)

# Dataset
sim.dat <- data.frame(time = times, status = f.status)


sim.dat <- cbind(sim.dat, X)

colnames(sim.dat)[3:22] <- paste0("X", seq_len(p))

head(sim.dat)
```

```
       time status         X1         X2         X3         X4         X5
1 0.9732177      2 -1.1628247  0.0321905  1.4222761 -0.8186584 -1.4009204
2 0.6872912      0  0.1455104  0.6884258 -1.5703404 -1.1555643 -1.4455326
3 0.9663264      0 -0.8014504  0.5858673 -0.8622731 -0.1846448 -0.2818599
4 0.5203338      1 -0.5265595 -0.1354980 -1.2839066  0.8681428 -1.1788005
5 0.9360684      1  0.3295775  1.3900472  0.7740371 -1.3138496  0.5637925
6 2.0669813      2  0.4856713  0.5122496  1.6680761  1.9106700 -0.3368238
          X6         X7          X8          X9         X10        X11
1  0.5675513 -0.4703791  0.69089707  0.02646950  0.08383588 0.78089840
2 -0.1463352  0.9886052  0.06755829  0.54770223  1.02899269 0.29398145
3 -0.1854775  1.4128005  0.67466436 -1.46226309 -0.64414266 1.13345275
4  0.1800869 -2.4742049 -0.21163409  1.80194613  0.67178223 0.06812753
5 -0.6002850  0.4885820 -1.01554200  0.07993189 -0.93527923 0.29881254
6  0.6669663  0.2683424  0.76820932 -0.90374743 -1.59158730 0.29233289
         X12         X13         X14        X15         X16        X17
1  1.6991617  1.90673935  0.43882008  1.4329088 -0.04673089 -0.1932456
2  0.8326698  0.04000960  0.68287200 -1.1468513 -0.07262573  0.7643219
3 -2.5446269 -0.45657949  0.01494344 -1.0946496 -0.14146290 -0.3432013
```

```
4 -0.6352412  0.11156271 -0.75627648 -0.2768998 -0.16981391  1.3208010
5 -0.5238516 -0.89814694 -1.12389866 -0.8977426  0.21538180  0.9471801
6 -0.3331147  0.03002175 -0.76667306 -1.5308352  0.55250380 -1.1552208
         X18         X19        X20
1  1.7609912  0.01477822 -1.1391273
2  0.1598004  0.76261396  1.5152245
3 -1.3382217 -0.42072239 -1.6095942
4 -0.9187628 -1.41869015  1.3853434
5  0.5087313  1.20650682 -0.4942597
6 -3.1051466 -0.33937759 -1.3164746
```

2. Floating precision of mtool estimators

```
# Split into training and test sets
train.index <- caret::createDataPartition(sim.dat$status, p = 0.70, list = FALSE)
train <- sim.dat[train.index,]
validation <- sim.dat[-train.index,]

surv_obj_train <- with(train, Surv(time, as.numeric(status), type = "mstate"))

cov_train <- cbind(train[3:22])

# Create case-base dataset
cb_data_train <- create_cbDataset(surv_obj_train, as.matrix(cov_train))

# Apply to validation set
# First fit
lambdagrid <- rep(0.01, 150)
cvs_res <- mclapply(lambdagrid, function(lambda_val) {
fit_val1 <- fit_cbmodel(cb_data_train, regularization = 'elastic-net',
                                         lambda = lambda_val , alpha = 1)
 }, mc.cores = 4)

lambdagrid <- rep(0.009, 150)
cvs_res1 <- mclapply(lambdagrid, function(lambda_val) {
fit_val1 <- fit_cbmodel(cb_data_train, regularization = 'elastic-net',
                                         lambda = lambda_val , alpha = 1)
 }, mc.cores = 4)
```

```r
non_zero_coefs <-  unlist(mclapply(cvs_res,
function(x) {return(x$no_non_zero)}, mc.cores = 4))

non_zero_coefs1 <-  unlist(mclapply(cvs_res1,
function(x) {return(x$no_non_zero)}, mc.cores = 4))

# Display different outputs
cvs_res[[1]]
```

```
$coefficients
22 x 2 sparse Matrix of class "dgCMatrix"

 [1,]  0.06838507  0.07947706
 [2,]  .           .
 [3,]  .           .
 [4,]  .           .
 [5,]  .           .
 [6,]  .           .
 [7,]  .           .
 [8,]  .           .
 [9,]  .           .
[10,]  .           .
[11,]  .           .
[12,]  .           .
[13,]  .           .
[14,]  .           .
[15,]  .           .
[16,]  .           .
[17,]  .           .
[18,]  .           .
[19,]  .           .
[20,]  0.09599489  0.03688073
[21,]  0.29352645  0.28722835
[22,] -0.12420916 -0.32630104

$no_non_zero
[1] 8
```

```r
cvs_res[[which(non_zero_coefs != 8)[1]]]
```

```
$coefficients
22 x 2 sparse Matrix of class "dgCMatrix"

 [1,]  6.839405e-02  0.07945370
 [2,]  .             .
 [3,]  .             .
 [4,]  .             .
 [5,]  .             .
 [6,]  .             .
 [7,]  .             .
 [8,]  .             .
 [9,]  .             .
[10,]  .             .
[11,]  .             .
[12,]  .             .
[13,]  .             .
[14,]  .             .
[15,]  .             .
[16,]  .             .
[17,]  2.433605e-08  .
[18,]  .             .
[19,]  .             .
[20,]  9.595715e-02  0.03685856
[21,]  2.934338e-01  0.28712829
[22,] -1.242562e-01 -0.32633242

$no_non_zero
[1] 9
```

```
prop.table(table(non_zero_coefs))
```

```
non_zero_coefs
          8           9          10          11          12 
0.633333333 0.226666667 0.113333333 0.020000000 0.006666667 
```

```
prop.table(table(non_zero_coefs1))
```

```
non_zero_coefs1
          8           9          10          11          12          13 
0.360000000 0.333333333 0.186666667 0.066666667 0.046666667 0.006666667 
```

Fixing the issue

Increased the `tolerance` from 1e-4 to 1e-5 and the `niter_inner_mtplyr` to 15 (from 5) (number of stochastic updates used to estimate the full gradient)

```
# Penalized Multinomial Logistic Regression
mtool.MNlogistic <- function(X, Y, offset, N_covariates,
                             regularization = 'l1', transpose = F,
                             lambda1, lambda2 = 0, lambda3 = 0,
                             learning_rate = 1e-4, tolerance = 1e-5,
                             niter_inner_mtplyr = 15, maxit = 100, ncores = -1,
```

Running the same test again with new function

```
lambdagrid <- rep(0.02, 150)
cvs_res <- mclapply(lambdagrid, function(lambda_val) {
fit_val1 <- fit_cbmodel(cb_data_train, regularization = 'elastic-net',
                                          lambda = lambda_val , alpha = 1)
 }, mc.cores = 4)

non_zero_coefs <-  unlist(mclapply(cvs_res,
function(x) {return(x$no_non_zero)}, mc.cores = 4))

prop.table(table(non_zero_coefs))
```

```
non_zero_coefs
6
1
```

```
lambdagrid <- rep(0.009, 150)
cvs_res1 <- mclapply(lambdagrid, function(lambda_val) {
fit_val1 <- fit_cbmodel(cb_data_train, regularization = 'elastic-net',
                                          lambda = lambda_val , alpha = 1)
 }, mc.cores = 4)

non_zero_coefs1 <-  unlist(mclapply(cvs_res1,
function(x) {return(x$no_non_zero)}, mc.cores = 4))

prop.table(table(non_zero_coefs1))
```

```
non_zero_coefs1
8
1
```

3. Re-writing cross-validation function

```
# Cross-validation function for mtool
# Implement brier score as a metric as well\C-index?
mtool.multinom.cv <- function(cb_data_train, regularization = 'elastic-net', lambdagrid =
constant_covariates = 2, n.cores = detectCores()-4, initial_max_grid = NULL, precision = 0
  # Default lambda grid
  if(is.null(lambdagrid)) {
    # Lambda max grid for bisection search
    if(is.null(initial_max_grid)) {
    initial_max_grid <-
  c(0.9, 0.5, 0.1, 0.07, 0.05, 0.01, 0.009, 0.005)
    fit_val_max <-  mclapply(initial_max_grid,
  function(lambda_val) {
    fit_cbmodel(cb_data_train, regularization = 'elastic-net',
                              lambda = lambda_val, alpha = alpha)}, mc.cores = n.cores)
    non_zero_coefs <-  unlist(mclapply(fit_val_max, function(x) {return(x$no_non_zero)}, m
   if(!isTRUE(any(non_zero_coefs == (constant_covariates*2)))){
     warning("Non-zero coef value not found in default grid. Re-run function and specify i
    }
    upper <- initial_max_grid[which(non_zero_coefs > (constant_covariates*2 + 1))[1]-1]
    lower <- initial_max_grid[which(non_zero_coefs > (constant_covariates*2 + 1))[1]]
    new_max_searchgrid <- seq(lower, upper, precision)
    fit_val_max <-  mclapply(new_max_searchgrid,
        function(lambda_val) {
   fit_cbmodel(cb_data_train, regularization = 'elastic-net',
      lambda = lambda_val, alpha = alpha)}, mc.cores = n.cores)
  non_zero_coefs <-  unlist(mclapply(fit_val_max, function(x) {return(x$no_non_zero)}, mc.
  lambda_max <- new_max_searchgrid[which.min(non_zero_coefs)]
  epsilon <- epsilon
    K <- grid_size
  lambdagrid <- rev(round(exp(seq(log(lambda_max), log(lambda_max*epsilon), length.out = K
    }
  }
  cb_data_train <- as.data.frame(cb_data_train)
  # Create folds
  folds <- caret::createFolds(factor(cb_data_train$event_ind), k = nfold, list = FALSE)
```

```
lambda.min <- rep(NA_real_, nfold)
all_deviances <- matrix(NA_real_, nrow = length(lambdagrid), ncol = nfold)
non_zero_coefs <- matrix(NA_real_, nrow = length(lambdagrid), ncol = nfold)

#Perform 10 fold cross validation
for(i in 1:nfold) {
  #Segment your data by fold using the which() function
  train_cv <- cb_data_train[which(folds != i), ] #Set the training set
  test_cv <- cb_data_train[which(folds == i), ] #Set the validation set
   # Create X and Y
  X <- as.matrix(cbind(train_cv[, grepl("covariates", names(train_cv))], train_cv$time))
  Y <- train_cv$event_ind
cvs_res <- mclapply(lambdagrid, function(lambda_val) {
  lambda1 <- lambda_val*alpha
  lambda2 <- 0.5*lambda_val*(1 - alpha)
  # mtool.MNlogistic is too verbose...
  mtool::mtool.MNlogistic(
    X = as.matrix(X),
    Y = Y,
    offset = train_cv$offset,
    N_covariates = constant_covariates,
    regularization = 'elastic-net',
    transpose = FALSE,
    lambda1 = lambda1, lambda2 = lambda2,
    lambda3 = 0)
  }, mc.cores = n.cores)
test_cv <- list("time" = test_cv$time,
               "event_ind" = test_cv$event_ind,
               "covariates" = test_cv[, grepl("covariates", names(test_cv))],
               "offset" = test_cv$offset)
mult_deviance <- unlist(lapply(cvs_res, multi_deviance, cb_data = test_cv))
all_deviances[, i] <- mult_deviance
mean_dev <- rowMeans(all_deviances)
lambda.min <- lambdagrid[which.min(mean_dev)]
cv_se <- sd(all_deviances[which(lambdagrid == lambda.min),])/sqrt(nfold)
dev.1se <- mean_dev[which.min(mean_dev)] + cv_se
lambda.1se <- lambdagrid[which((mean_dev <= dev.1se))]
lambda.1se <- tail(lambda.1se, n = 1)
non_zero_coefs[, i] <-  unlist(mclapply(cvs_res, function(x) {return(x$no_non_zero)}, mc
rownames(all_deviances) <- lambdagrid
rownames(non_zero_coefs) <- lambdagrid
```

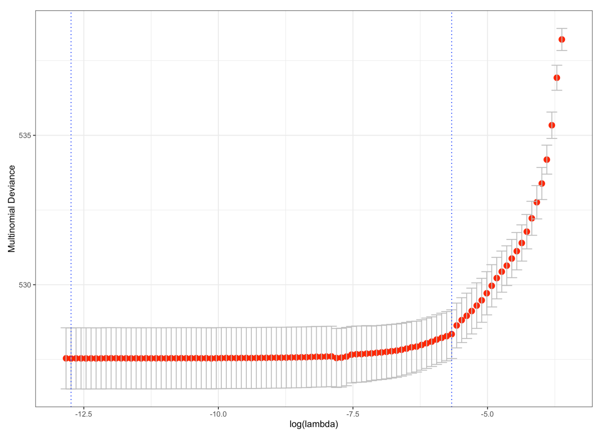
```
  }
  if(isTRUE(plot)){
    row_stdev <- apply(all_deviances, 1, function(x) {sd(x)/sqrt(nfold)})
    plot.dat <- data.frame(lambdagrid = lambdagrid, mean.dev = mean_dev,
      upper = mean_dev +row_stdev, lower = mean_dev - row_stdev)
   p <- ggplot2::ggplot(plot.dat, aes(log(lambdagrid), mean.dev)) + geom_point(colour = "r
  }
  return(list(lambda.min = lambda.min,  non_zero_coefs = non_zero_coefs,
              lambda.1se = lambda.1se, cv.se = cv_se, plot.cv = p))
}
```

```
fit.cv <- mtool.multinom.cv(cb_data_train, alpha = 1)
```

```
fit.cv$lambda.min
fit.cv$lambda.1se
```

[1] 2.9413e-06

[1] 0.0034613531



4. Final competitor models for casebase

Link to paper

Final list

1. Binomial model - Implmeneted
2. CoxBoost (Boosted Fine-Gray) - Implemented

3. Penalized Fine-Gray - Implemented
4. Quantile regression with pseudo values - To implement

5. To do and what I am working on now

1. Final functions for CIF and Brier score (adapted to competing risks)
2. Weights for the casebase models?